A Theory of Information-Flow Labels

Benoît Montagu
University of Pennsylvania
Philadelphia, USA

Benjamin C. Pierce
University of Pennsylvania
Philadelphia, USA

Randy Pollack
Harvard University
Cambridge, USA

***Abstract*—The security literature offers a multitude of calculi, languages, and systems for information-flow control, each with some set of *labels* encoding security policies that can be attached to data and computations. The exact form of these labels varies widely, with different systems offering many different combinations of features addressing issues such as confidentiality, integrity, and policy ownership. This variation makes it difficult to compare the expressive power of different information-flow frameworks.**

To enable such comparisons, we introduce *label algebras*, an abstract interface for information-flow labels equipped with a notion of *authority*, and study several notions of *embedding* between them. The simplest is a straightforward notion of *injection* between label algebras, but this lacks a clear computational motivation and rejects some reasonable encodings between label models. We obtain a more refined account by defining a space of encodings parameterized by an interpretation of labels and authorities, thus giving a semantic flavor to the definition of encoding. We study the theory of semantic encodings and consider two specific instances, one based on the possible observations of boolean values and one based on the behavior of programs in a small lambda-calculus parameterized over an arbitrary label algebra.

We use this framework to define and compare a number of concrete label algebras, including realizations of the familiar *taint*, *endorsement*, *readers*, and *distrust* models, as well as label algebras based on several existing programming languages and operating systems.

***Keywords*-Security; Languages; Design; Theory; Information flow control (IFC); DIFC; label models; decentralized label model (DLM); JIF; LIO; disjunction category model; Flume; HiStar; Asbestos.**

I. Introduction

Information-flow control (IFC) systems [1] run the gamut from static type systems to run-time monitors and from core calculi to full-blown languages and operating systems. A critical component of each one is a *label model*—a notation for writing down information-flow labels together with rules for when one label *flows to* another, in the sense that data labeled with the first is allowed to flow to contexts labeled with the second. These labels can be thought of as low-level "micro-policies" for information flow. They do not directly describe the end-to-end security policies that the system's users might care about ("my banking information will never be sent to `evil.com`"); rather, they capture information-flow invariants on specific sensitive values ("this integer and values derived from it should only be visible to the *Bank* principal"), which can be used by programmers to enforce and reason about higher-level security properties.

Label models come in a bewildering variety of shapes and forms. A theoretical discussion of IFC might typically use a very simple model with just two or three labels ($\bot \sqsubseteq \top$ or $public \sqsubseteq secret \sqsubseteq topsecret$), or else assume an arbitrary lattice of labels. Or a label could be defined to be a set of *principals*, interpreted either as the set of entities that are allowed to read a given value or as the set of entities that trust or endorse it, or perhaps as the set of entities that may have tainted it. More complex systems use sets of sets, logical formulae, or other structures as labels. Some systems—e.g., those based on the Decentralized Label Model (DLM) [2]—include a notion of *policy owners*, distinct from readers, tainters, or endorsers, enabling programmers to control not only who can *use* labeled values but also who can *change* their labels by declassification. Some systems focus on protecting secrecy, others integrity, and still others incorporate both. The list of possible variations seems endless.

Many questions now arise. Which label models are best for which purposes? What common structure might we expect *every* label model to have, beyond a simple flows-to ordering? Can all the label models used in real systems be viewed as instances of this common structure, or are there deep differences between them? What generic operations can be performed on arbitrary label models? Can the common dictum that "integrity is formally dual to secrecy" be given a rigorous explanation? Is some label model $\mathcal{M}_1$ "more expressive" than a model $\mathcal{M}_2$, in the sense that, given a program written in terms of $\mathcal{M}_1$, we can obtain a program over $\mathcal{M}_2$ (with the same behavior!) by rewriting labels in some systematic way? Such questions are rarely discussed, but they seem essential to a thorough understanding of IFC.

Our goal in this paper is to initiate the comparative study of label models by providing a concrete mathematical framework and investigating how it applies to label models found in the wild. We define *label algebras*, an abstract presentation of the mechanisms of information flow and authority common to many label models. On top of this, we define a simple programming language—an untyped lambda-calculus with dynamic information-flow tracking and declassification—parameterized by an arbitrary label algebra (§III). We give a generic proof of (an authority-enriched generalization of) a standard non-interference property.

From the definition of label algebras, we directly obtain an algebraic notion of *injections*—maps that preserve and reflect the structure of label algebras—as a natural, algebraically justified way of formalizing claims of the form "label algebra $\mathcal{M}_1$ can be faithfully encoded in $\mathcal{M}_2$." However, this notion of encoding is quite strong—so strong that some intuitively reasonable embeddings fail to be injections. (For example, the label algebra of conjunctions of literals cannot be injected into the label algebra of conjunctions of disjunctions of literals, although the latter seems intuitively "more expressive" than the former.) Moreover, we would like our notion of embedding to have some *semantic* justification. We therefore introduce a generic notion of *semantic embeddings* between label algebras (§IV), parameterized by the choice of "semantics": different semantics may lead to different notions of embeddings. We study embeddings for two different semantics: a *boolean semantics*, which focuses on an observer's possible observations of labeled boolean values in a given label algebra, and an *evaluation semantics*, which additionally takes into account the behavior of computations over labeled data. For each of them, we derive an algebraic characterization theorem that can be used to verify the existence or nonexistence of embeddings.

Finally, we use these concepts to define and study a number of concrete label algebras, including simple examples that illustrate dimensions of the design space of label models, realizations of the familiar *taint*, *endorsement*, *readers*, and *distrust* models (§V), and more complex examples based on real-world languages and operating systems (§VI). In particular, we define label algebras corresponding to the *disjunction category* (DC) model [3], the Decentralized Label Model (DLM) [2] without principal hierarchies, Asbestos [4], HiStar [5], [6], and Laminar [7]; we also discuss Flume [8] and show that its labels are a little more flexible than what label algebras can represent. We settle the existence or non-existence of embeddings among all of the simple examples and some of the real-world ones—in particular, we show that (the secrecy part of) the DLM with no principal hierarchy cannot be embedded in the DC model (we conjecture the converse embedding is impossible too), but that, when authorities are not considered, embeddings between the underlying label lattices do exist in both directions. Finally, we discuss models with principal hierarchies (such as the full DLM [2]) and show how these can be modeled as a component of the authority structure of a label algebra (§VII). A known order-theoretic weakness of the DLM with a principal hierarchy prevents it from satisfying all the requirements to be a label model, but we can complete its order structure to yield a label algebra with nearly the same behavior.

We survey related work in §VIII and sketch directions for future work in §IX.

Two important caveats should be mentioned at the outset. First, our definition of label algebra covers just a small set of core features—labels, label ordering, authority, and defaults—omitting some of the interesting complexities associated with real-world label models. In particular, this paper does not address dynamic generation of principals and authorities, although we briefly sketch an extension of label algebras that handles this feature in §IX. Second, since evaluation embeddings are defined in terms of computation, their properties necessarily depend on the details of the programming language under consideration; adding other features such as first-class labels will change some of our results. This specificity is inherent to our approach; indeed, we show that injections are the only form of encoding that is system independent.

We have verified most of our theorems with the Coq proof assistant [9] (these theorems are labeled with the symbol 🐓). The full Coq development can be found at http://www.cis.upenn.edu/~bcpierce/papers.

II. Label Algebras

Basic definitions. Recall that a pre-lattice is a preorder with meet and join operations. Note that there may be cycles in a preorder ($x \leq y$ and $y \leq x$ with $x \neq y$).

II.1 Definition: A *label algebra* $\mathcal{M}$ comprises:

- a pre-lattice of *labels* $(\mathcal{L}, \sqsubseteq, \sqcap, \sqcup)$
- a lower-bounded join pre-semilattice of *authorities* $(\mathcal{A}, \leq, \vee, 0)$,
- for each authority A, a *flows-to* relation $\sqsubseteq_A$ on $\mathcal{L}$, such that:
  1) $\sqsubseteq_0 = \sqsubseteq$
  2) if $A \leq A'$ and $L_1 \sqsubseteq_A L_2$, then $L_1 \sqsubseteq_{A'} L_2$
  3) each $(\mathcal{L}, \sqsubseteq_A, \sqcap, \sqcup)$ is a pre-lattice
- a designated *default label* L^{def}.

We write $L_1 \equiv_A L_2$ when $L_1 \sqsubseteq_A L_2$ and $L_2 \sqsubseteq_A L_1$; we write $\equiv$ to denote $\equiv_0$; we also write $A_1 \equiv A_2$ when $A_1 \leq A_2$ and $A_2 \leq A_1$. We write $\mathcal{LA}$ for the set of label algebras.

The main structure in a label algebra is the set of labels, which must form a pre-lattice—i.e. labels must be equipped with a pre-order $\sqsubseteq$ and a greatest lower bound $\sqcap$ and a least upper bound $\sqcup$.

Authorities can be understood as permissions to bend the rules of information flow, allowing more flows between labels. The least (or empty) authority, written 0, carries no privilege: the relation $\sqsubseteq_0$ is exactly the flows-to relation of the underlying pre-lattice of labels (axiom 1 about authorities).

The authority-indexed flows-to relations are compatible with the ordering on authorities (axiom 2): increasing authority makes it easier to flow from one label to another. *Declassification* (or *downgrading*) is the exercise of authority to permit a flow that would not otherwise be allowed. The 0-authority flows-to relation describes the flows that are always allowed.

Axiom 3 requires that all the pre-lattices in the family have the same joins and meets—i.e. joins and meets don't depend on authority. Remember that in a pre-lattice, joins and meets are unique up to equivalence. Axiom 3 is consistent with the fact that information flow analyses combine labels in a way that is independent of the authority that a piece of program could use.

Many label models have distinguished bottom and top labels, but we do not ask the label pre-lattices to be bounded.

The last bullet in the definition specifies that there should be a designated *default label*. That label could be used, for instance, to annotate all data values by default, unless they have been downgraded. This is useful when labels are used for *endorsement*: by default, a data value starts out life endorsed by no one (i.e., with a high label), and its label gets lowered only by the explicit exercise of authority. In the evaluation semantics defined in §IV, this is achieved by using the default label as the starting value of the *pc label* when a term is evaluated.

Examples. We will see many examples of label algebras in §V and §VI; for now, let's look at just a few simple ones.

Most label algebras are defined over some enumerable set of *principals*, written $\mathbb{P}$. We write p for specific principals, P for sets of principals, and $\mathcal{P}(\mathbb{P})$ for the set of sets of principals. It is sometimes convenient to consider $\mathcal{P}(\mathbb{P})$ as a lattice, with intersection and union corresponding to meet and join. Similarly, $\mathcal{P}^{fin}(\mathbb{P})$ is the set or lattice whose elements are finite sets of principals. $\mathbf{1}$ is the unit lattice; its single element is written either $\bot$ or 0.

One very simple label algebra is the *public / secret* model, which we call **2** for short; it is also sometimes called the *binary* model [10]. Its set of labels is a two-point lattice, and it has only one authority.

2: Public / Secret Model

$$\mathcal{L} = \{\bot, \top\} \qquad \bot \sqsubseteq \top \qquad L^{def} = \bot \qquad \mathcal{A} = \mathbf{1}$$

A more interesting label algebra is the *readers model* (written **CR**, rather than just **R**, for consistency with a group of related label algebras that we will encounter in §V).

CR: Readers Model

$$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad L^{def} = \mathbb{P}$$
$$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad A_1 \leq A_2 = A_1 \subseteq A_2$$
$$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2$$

Its labels are either the full set of principals or one of its finite subsets, ordered by reverse inclusion. Intuitively, the principals in a label are the ones who *may read* some piece of data. Its default label is $\mathbb{P}$ (which is the bottom element of the label lattice)—i.e. anybody is allowed to read data with the default label. A value labeled with some set of principals can freely be relabeled with a smaller set (fewer allowed readers)—in particular, in the labeled lambda-calculus in §III, it will always be legal to take a value with the default label and relabel it (restrict its readership) to some finite set of principals. Authorities are sets of principals, and an authority containing a principal p permits flows from labels not including p (i.e. data that p cannot read) to labels where p is allowed as a reader. For example, it is legal to relabel a value labeled $\{q, r\}$ into one labeled $\{p, q, r\}$ using the authority $\{p, s\}$. The top element of the authority lattice, $\mathbb{P}$, is an omnipotent authority: it allows any flow whatsoever. We do not expect it to be used by any actual program or observer.

Another simple label algebra is the *endorsement model* (**CE**). It differs from the readers model only in its default label, which is the top element. The principals in a label indicate who has *endorsed* some data value. By default, nobody endorses anything.

CE: Endorsement Model

$$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad L^{def} = \emptyset$$
$$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\} \qquad A_1 \leq A_2 = A_1 \subseteq A_2$$
$$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2$$

Operations on label algebras. The space of label algebras is closed under some simple operations, including dualization and product; these can be useful for describing examples compactly.

Suppose $\mathcal{M}$ is a label algebra. Its *dual*, $\mathcal{M}^{op}$, is obtained by reversing the $\sqsubseteq_A$ relations:

$\mathcal{M}^{op}$: Dual of $\mathcal{M}$

$$\mathcal{L}^{op} = \mathcal{L} \qquad \mathcal{A}^{op} = \mathcal{A}$$
$$L_1 \sqsubseteq^{op} L_2 = L_2 \sqsubseteq L_1 \qquad \sqcup^{op} = \sqcap \qquad \sqcap^{op} = \sqcup$$
$$L_1 \sqsubseteq_A^{op} L_2 = L_2 \sqsubseteq_A L_1 \qquad (L^{def})^{op} = L^{def}$$

Because authorities have no top element in general, we do not invert the authority structure (there would be no canonical way of choosing a bottom authority). Moreover, because there is no canonical "complement" for the default element, we keep it the same.

Suppose $\mathcal{M}_1$ and $\mathcal{M}_2$ are two label algebras. We define their product $\mathcal{M}_1 \times \mathcal{M}_2$ as follows:

$\mathcal{M}_1 \times \mathcal{M}_2$: Product of $\mathcal{M}_1$ and $\mathcal{M}_2$

$$\mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2 \qquad \mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \qquad L^{def} = (L_1^{def}, L_2^{def})$$
$$(L_1, L_2) \sqsubseteq_{(A_1, A_2)} (L_1', L_2') = (L_1 \sqsubseteq_{A_1} L_1' \wedge L_2 \sqsubseteq_{A_2} L_2')$$

Real world systems often combine secrecy and integrity into a single model by taking a label algebra of the form $\mathcal{M} \times \mathcal{M}^{op}$ for some $\mathcal{M}$ (see §VI).

We can also drop the authority part of an arbitrary label algebra. (We will use this operation several times in §V and

§VI.) Given a label algebra $\mathcal{M}$, its 0-*authority projection*, written $\mathcal{M}^{\mathbf{0}}$, is defined as follows:

$\mathcal{M}^{\mathbf{0}}$: 0-authority projection of $\mathcal{M}$

$$\mathcal{L}^{\mathbf{0}} = \mathcal{L} \quad \mathcal{A}^{\mathbf{0}} = \mathbf{1} \quad L_1 \sqsubseteq^{\mathbf{0}}_A L_2 = L_1 \sqsubseteq L_2 \quad L^{\mathit{def}\mathbf{0}} = L^{\mathit{def}}$$

Label algebra morphisms and injections. We compare the expressiveness of label algebras by studying the absence or existence or certain kinds of maps between them. We begin with a very loose notion of maps and then consider *morphisms*—i.e. structure-preserving maps—and *injections*—i.e. structure-reflecting morphisms. Later (§IV), we define embeddings—i.e. semantics-preserving and reflecting maps—which will be our real objects of study. We will see that any injection is an embedding (Theorem IV.7), but that embeddings are not necessarily morphisms (Theorems IV.8 and IV.10).

II.2 Definition: Given two label algebras $\mathcal{M}_1 = (\mathcal{L}_1, \mathcal{A}_1, \ldots)$ and $\mathcal{M}_2 = (\mathcal{L}_2, \mathcal{A}_2, \ldots)$, a *label algebra map* m from $\mathcal{M}_1$ to $\mathcal{M}_2$, written $m \in \mathcal{M}_1 \to \mathcal{M}_2$, is a pair of:

- a function (also written m) from $\mathcal{L}_1$ to $\mathcal{L}_2$, and
- a function (also written m) from $\mathcal{A}_1$ to $\mathcal{A}_2$.

II.3 Definition [Morphism]: A map $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is a morphism when:

- $L_1 \sqsubseteq_A L_2$ implies $m(L_1) \sqsubseteq_{m(A)} m(L_2)$
- $m(L_1^{\mathit{def}}) \equiv L_2^{\mathit{def}}$
- $L_1 \equiv L_2$ implies $m(L_1) \equiv m(L_2)$
- $m(L_1 \sqcup L_2) \equiv m(L_1) \sqcup m(L_2)$
- $m(L_1 \sqcap L_2) \equiv m(L_1) \sqcap m(L_2)$
- $m(0_1) \equiv 0_2$
- $A_1 \equiv A_2$ implies $m(A_1) \equiv m(A_2)$
- $m(A_1 \vee A_2) \equiv m(A_1) \vee m(A_2)$

A morphism is a structure-preserving map: it is monotone, preserves the default labels and the bottom authority, commutes with joins and meets, and preserves equivalence classes. We have chosen the most natural definition for a structure preserving map. One can notice however that some items in the definition are redundant. For instance, item 3 is implied by the conjunction of item 1, item 7 and axiom 1 of label algebras.

II.4 Definition [Injection]: A morphism $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is an *injection* when:

- $m(L_1) \sqsubseteq_{m(A)} m(L_2)$ implies $L_1 \sqsubseteq_A L_2$;
- $m(A_1) \equiv m(A_2)$ implies $A_1 \equiv A_2$;
- $m(L_1) \equiv m(L_2)$ implies $L_1 \equiv L_2$.

In that case, we write $m \in \mathcal{M}_1 \hookrightarrow \mathcal{M}_2$. We write $\mathcal{M}_1 \hookrightarrow \mathcal{M}_2$ if there exists $m \in \mathcal{M}_1 \hookrightarrow \mathcal{M}_2$, and we say that $\mathcal{M}_1$ *injects into* $\mathcal{M}_2$.

Injections are morphisms that also *reflect* the structure: they reflect the authority-indexed flows-to relation and the equivalence classes. Note that injections also necessarily reflect the orderings on labels and on authorities. Our notion of injection is reminiscent of the notion of *order embedding* (i.e. order-preserving and reflecting functions).

II.5 Proposition: The relation $\hookrightarrow$ between label algebras is reflexive and transitive.

Given the strong algebraic flavor of injections, one could consider that they would make a good definition of embedding. It turns out that they are too constraining: they ask for the *whole* structure to be preserved and reflected, whereas a given information-flow system or language may use the flows-to relation only (as in DStar [11]), while another one may additionally use joins but make no use of meets (like the toy λ-calculus of §III), and yet another may use both meets and joins (like Jif [2]). In these cases, why should embeddings talk about the label algebra ingredients that a system does not use? The next two sections develop a more refined notion that takes these issues into account.

$$
\begin{array}{lcl}
b & ::= & \mathit{tt} \mid \mathit{ff} \qquad \text{(booleans)} \\
t & ::= & b \mid x \mid \lambda x.t \mid t\ t \mid t \updownarrow_A L \qquad \text{(terms)} \\
v & ::= & b \mid \langle \rho, \lambda x.\ t \rangle \qquad \text{(values)} \\
a & ::= & v@L \qquad \text{(atoms)} \\
\rho & ::= & \bullet \mid \rho, x = a \qquad \text{(environments)}
\end{array}
$$

Figure 1. Syntax of terms, values, atoms and environments.

III. Labeled Lambda-Calculus

We now define a small programming language $\lambda_{\mathcal{M}}$ parameterized by a label algebra $\mathcal{M}$. For simplicity, we choose an untyped language with dynamic information-flow tracking, similar to that of [12].

Syntax and semantics. The syntax of $\lambda_{\mathcal{M}}$, together with the sets of values and labeled values (atoms), is shown in Figure 1. Its syntax comprises booleans, variables, λ-abstractions, applications and a construct $t \updownarrow_A L$ to *relabel* the result of the evaluation of t with the constant label L using authority A. Note that in this tiny language labels and authorities are not first-class: they can only occur in the relabeling construct.

The fact that the language $\lambda_{\mathcal{M}}$ is so small and simple is a deliberate choice: for example, there is no construct that controls who can exercise which authority. While it may look unrealistic, this modeling simplification permits to easily define the static authority of a program (Definition III.1).

The big-step operational semantics of $\lambda_{\mathcal{M}}$ is given in Figure 2. The evaluation judgment has the form $pc, \rho \vdash t \Downarrow a$. Evaluation produces *atoms*, denoted by a, which are labeled values. We write $v@L$ to denote the atom whose value component is v and whose label is L. The environment ρ maps variables to closed atoms.

The label pc is the *program counter* label; as usual, it tracks implicit flows of information through the control state of the

$$\frac{}{pc, \rho \vdash tt \Downarrow tt@pc} \quad \textsc{Eval\_True}$$

$$\frac{}{pc, \rho \vdash ff \Downarrow ff@pc} \quad \textsc{Eval\_False}$$

$$\frac{\rho(x) = v@L}{pc, \rho \vdash x \Downarrow v@(pc \sqcup L)} \quad \textsc{Eval\_Var}$$

$$\frac{}{pc, \rho \vdash (\lambda x.t) \Downarrow \langle \rho, \lambda x.\, t \rangle @ pc} \quad \textsc{Eval\_Abs}$$

$$\frac{\begin{array}{l} pc, \rho \vdash t_1 \Downarrow \langle \rho_1, \lambda x.\, t \rangle @ L_1 \\ pc, \rho \vdash t_2 \Downarrow a_2 \\ L_1, (\rho_1, x = a_2) \vdash t \Downarrow a_3 \end{array}}{pc, \rho \vdash (t_1\ t_2) \Downarrow a_3} \quad \textsc{Eval\_App}$$

$$\frac{\begin{array}{l} pc, \rho \vdash t_1 \Downarrow v_1 @ L_1 \\ L_1 \sqsubseteq_A L_2 \end{array}}{pc, \rho \vdash t_1 \updownarrow_A L_2 \Downarrow v_1 @ L_2} \quad \textsc{Eval\_Relabel}$$

Figure 2. Big-step semantics.

program. The pc label is modified by branching over a secret value: in particular, the pc label may change when a function is called (third premise of rule EVAL_APP). If our language had other control constructs such as conditionals, they would need similar side conditions. In the variable lookup rule, the label on the variable's value from the environment is joined with the current pc label to form the label on the result, reflecting the fact that the choice to look up this variable (as opposed to another one, for example) may have been influenced by sensitive information. This detail is crucial for the non-interference theorem (III.3).

We can lift a label algebra map m to a function $\hat{m}$ on programs (resp. values, atoms, environments) as a term homomorphism (resp. values, etc.) that transforms label constants and authorities using m and copies everything else unchanged. The operation of lifting label algebra maps commutes with composition of maps, and transforms the identity map into the identity function.

Basic properties. We now establish some fundamental results about $\lambda_{\mathcal{M}}$—in particular, a standard non-interference property. This is mainly a sanity check on our labeled lambda-calculus: the only part of this development that is used in later sections is Definition III.2.

III.1 Definition: The *authority of a program* t, written $Auth(t)$, is the join of all the authorities that occur in t. The definition is lifted to atoms, values, and environments.

The relation $\approx_A^L$ expresses *indistinguishability* of booleans for observers at label L and authority A.

III.2 Definition: $b_1@L_1 \approx_A^L b_2@L_2$ iff either

- $L_1 \not\sqsubseteq_A L$ and $L_2 \not\sqsubseteq_A L$, or
- $L_1 \sqsubseteq_A L$ and $L_1 \equiv L_2$ and $b_1 = b_2$.

Note that this relation is an equivalence.

III.3 Theorem [Non-interference]: If

- $pc, \rho_1 \vdash t \Downarrow b_1@L_1$ and $pc, \rho_2 \vdash t \Downarrow b_2@L_2$,
- $\operatorname{dom} \rho_1 = \operatorname{dom} \rho_2$ and $\rho_1(x) \approx_A^L \rho_2(x)$ for every $x \in \operatorname{dom} \rho_1$,
- $Auth(t) \leq A$,

then $b_1@L_1 \approx_A^L b_2@L_2$.

A point to note about the non-interference theorem is the mention of authority in its third assumption: the theorem requires the observer to have at least the same authority as the one of the program that is observed. This assumption permits the observer to perform all the downgradings that the program could perform, and thus gives more precision to the observation. (Removing the assumption would falsify the theorem: the new statement would say that all programs would be non-interfering in the usual sense, even those which downgrade labels.)

IV. Semantics of Labels

Now we can return to the question of what it means to claim that "Label model $\mathcal{M}_2$ can encode label model $\mathcal{M}_1$". We can give at least two interpretations of this claim with a firm basis in semantics—that is, in some notion of *observation*. The first is based on the notion of observation of boolean values from Definition III.2; we call maps that preserve and reflect such observations *boolean embeddings*. The second is based on observing the results of evaluation of programs; we call maps that preserve and reflect these observations *evaluation embeddings*. The latter interpretation is probably the more interesting one, since it is ultimately the behavior of whole programs that we are concerned with, but the first is also worth studying because claims about encodability of one label model in another are often justified by appealing to a static embedding of labels, without reference to the behavior of programs using those labels.

It is technically convenient to derive both interpretations from a common framework. In this section we define a general notion of semantics for labels, from which arises a generic notion of *embedding*. We connect this semantic notion of embedding to the injections that we have seen earlier. Then, we present the boolean and evaluation embeddings as two instances of that general framework.

Basic definitions. We write $\mathcal{R}_1 \precsim \mathcal{R}_2$ when $\mathcal{R}_1$ is a coarser relation than $\mathcal{R}_2$—that is, when $\mathcal{R}_2 \subseteq \mathcal{R}_1$.

IV.1 Definition [Label semantics]: A *label semantics* is a label-algebra-indexed family of sets $\mathcal{X} = \{\mathcal{X}_{\mathcal{M}}\}_{\mathcal{M} \in \mathcal{LA}}$ together with a function

$$[\![\cdot]\!] : (\mathcal{M} : \mathcal{LA}) \to (\mathcal{A}_{\mathcal{M}} \times \mathcal{L}_{\mathcal{M}}) \to \mathsf{BinRel}(\mathcal{X}_{\mathcal{M}})$$

that maps each pair (A, L) of an authority and a label from the same label algebra $\mathcal{M}$ to a binary *indistinguishability*

relation on $\mathcal{X}_{\mathcal{M}}$. The notation $(\mathcal{M} : T) \to U$ denotes a dependent product. Since $\mathcal{M}$ can be recovered from A and L we elide the first argument and write $[\![(A, L)]\!]$ for $[\![\cdot]\!]\ \mathcal{M}\ (A, L)$.

Below (Definitions IV.3 and IV.4), we will define two label semantics—one a semantics of boolean atoms (where $\mathcal{X}_{\mathcal{M}}$ is the set of boolean atoms over $\mathcal{M}$), the other a semantics of programs (where $\mathcal{X}_{\mathcal{M}}$ is the set of programs over $\mathcal{M}$).

This definition of label semantics is quite loose; in what follows, we restrict ourselves to *good* semantics.

IV.2 Definition [Good semantics]: A label semantics $[\![\cdot]\!]$ is *good* if:

1) $A_1 \leq A_2$ implies $[\![(A_1, L)]\!] \precsim [\![(A_2, L)]\!]$
2) $L_1 \sqsubseteq_A L_2$ implies $[\![(A, L_1)]\!] \precsim [\![(A, L_2)]\!]$
3) $\mathcal{X}$ is equipped with an action of label algebra maps—i.e. for any $m \in \mathcal{M}_1 \to \mathcal{M}_2$, there is a function $\widehat{m} \in \mathcal{X}_{\mathcal{M}_1} \to \mathcal{X}_{\mathcal{M}_2}$ such that $\widehat{id} = id$ and $\widehat{m_1 \circ m_2} = \widehat{m_1} \circ \widehat{m_2}$
4) $[\![\cdot]\!]$ is invariant under injections, i.e. for any injection $m \in \mathcal{M}_1 \hookrightarrow \mathcal{M}_2$ and any x, y, A, and L, we have $(x, y) \in [\![(A, L)]\!]$ iff $(\widehat{m}(x), \widehat{m}(y)) \in [\![(m(A), m(L))]\!]$.

The first two points require that a semantics captures a notion of *observation*: the relation $[\![(A, L)]\!]$ describes an observer with authority A and clearance L, and its observations get finer when one raises its authority or its clearance. Point 3 asks that it makes sense to apply a label algebra map to the objects that are observed. This is crucial, as we will see later that our notion of embedding is based on the changes that the application of a map could produce on observations. The action of label algebra maps on objects will be the key ingredient for transitive reasoning on embeddings (Proposition IV.6). Point 4 requires the semantics not to distinguish two label algebras that are the same up to injections. Take for instance the label algebra **CR'** that is the same as **CR**, except that labels and authorities are lists of principals instead of finite sets. There is clearly an injection m from any of these two label algebras to the other one: they are indeed morally "the same". Point 4 rules out any semantics $[\![\cdot]\!]$ where $[\![(A, L)]\!]$ would be different from $[\![(m(A), m(L))]\!]$. We will see later that a consequence of point 4 is that injections necessarily preserve and reflect good semantics (that is one half of Theorem IV.7).

In the rest of the paper, we consider two semantics: the boolean semantics is exactly the indistinguishability relation used by the non-interference theorem (III.3); the evaluation semantics is a semantics of programs, and relates programs that lead to indistinguishable booleans when they are evaluated in the empty environment, starting with the default label.

IV.3 Definition [Boolean semantics]: The boolean semantics, written $[\![\cdot]\!]^{\mathrm{b}}$ is defined as follows: $[\![(A, L)]\!]^{\mathrm{b}} = \approx_A^L$.

IV.4 Definition [Evaluation semantics]: The evaluation semantics, written $[\![\cdot]\!]^{\mathrm{e}}$ is defined as follows: $[\![(A, L)]\!]^{\mathrm{e}} = \{(t_1, t_2) \mid L^{def}, \bullet \vdash t_1 \Downarrow b_1@L_1$ and $L^{def}, \bullet \vdash t_2 \Downarrow b_2@L_2$ and $b_1@L_1 \approx_A^L b_2@L_2\}$.

Note that the evaluation semantics is the first definition that makes use of the default label of label algebras.

Both the boolean semantics and the evaluation semantics are good (🐓). For the evaluation semantics, we picked a particular partial equivalence on programs: it is rather simple but already interesting. It is also natural to consider different semantics (e.g. closed under contexts, dealing with non-termination, trace-based…). We leave that for future work.

Label algebra embeddings. We now define some properties that characterize label algebra maps that behave well with respect to some semantics.

IV.5 Definition [Sound, Complete, Embedding]: Suppose $m \in \mathcal{M}_1 \to \mathcal{M}_2$ and $[\![\cdot]\!]$ is a label semantics.

- We say that m is *sound* with respect to $[\![\cdot]\!]$ if $(x, y) \in [\![(A, L)]\!]$ implies $(\hat{m}(x), \hat{m}(y)) \in [\![(m(A), m(L))]\!]$ for any A, L, x, and y.
- We say that m is *complete* with respect to $[\![\cdot]\!]$ if $(\hat{m}(x), \hat{m}(y)) \in [\![(m(A), m(L))]\!]$ implies $(x, y) \in [\![(A, L)]\!]$ for any A, L, x, and y.
- We call m an *embedding* with respect to $[\![\cdot]\!]$ (written $m \in \mathcal{M}_1 \overset{[\![\cdot]\!]}{\hookrightarrow} \mathcal{M}_2$) if it is both sound and complete.
- $\mathcal{M}_1$ *embeds in* $\mathcal{M}_2$ for the semantics $[\![\cdot]\!]$, (written $\mathcal{M}_1 \overset{[\![\cdot]\!]}{\hookrightarrow} \mathcal{M}_2$) if there exists a function m such that $m \in \mathcal{M}_1 \overset{[\![\cdot]\!]}{\hookrightarrow} \mathcal{M}_2$.

Intuitively, a sound map can only decrease the power of an observer to distinguish, whereas a complete map can only increase distinguishing power. Embeddings are the maps that do not change the power of the observer. We'll see that different label semantics lead to different kinds of embeddings.

For any good semantics, embeddability allows transitive reasoning, thanks to item 3 of definition IV.2.

IV.6 Proposition: For any good label semantics $[\![\cdot]\!]$, the relation $\overset{[\![\cdot]\!]}{\hookrightarrow}$ on label algebras is reflexive and transitive. 🐓

The notion of embedding is weaker than the notion of injection, and an injections are the "best" embeddings for good semantics.

IV.7 Theorem: Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two label algebras. $\mathcal{M}_1 \hookrightarrow \mathcal{M}_2$ iff for any good semantics $[\![\cdot]\!]$, $\mathcal{M}_1 \overset{[\![\cdot]\!]}{\hookrightarrow} \mathcal{M}_2$. 🐓

This theorem explains that injections form the most precise notion of encoding: embeddings are less precise, and the loss of information is specified by the semantics one has chosen. The proof of that theorem relies on item 4 of definition IV.2

for one way of the implication; the other way is proved by defining a good semantics for which embeddings are necessarily injections.

Directly proving or refuting embeddability statements between label algebras under a given semantics can be a difficult task. To make it easier, the next two sections are devoted to establishing algebraic characterizations for boolean-embeddability and evaluation-embeddability.

Boolean embeddings. We write $\overset{\text{b}}{\hookrightarrow}$ to denote embeddability with respect to the boolean semantics.

IV.8 Theorem [Characterization of $\overset{\text{b}}{\hookrightarrow}$]: A label algebra map $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is a boolean embedding iff for any $L_1, L_2 \in \mathcal{L}_1$ and $A \in \mathcal{A}_1$:

1) $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$;
2) $L_1 \equiv L_2$ iff $m(L_1) \equiv m(L_2)$. 🐓

Note that the second item is not implied by the first since we know nothing about the image of the bottom authority. Also, note that the default label does not occur in the characterization of boolean embeddings since it is not used in the definition of the boolean semantics. Thus, boolean embeddings enjoy the following property.

IV.9 Proposition: Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two label algebras differing only in their default labels. Then, $\mathcal{M}_1 \overset{\text{b}}{\hookrightarrow} \mathcal{M}_2$. 🐓

Evaluation embeddings. We write $\overset{\text{e}}{\hookrightarrow}$ to denote embeddability with respect to the evaluation semantics.

IV.10 Theorem [Characterization of $\overset{\text{e}}{\hookrightarrow}$]: A map $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is an evaluation embedding iff for any $A, A' \in \mathcal{A}_1$, and $L, L_1, L_2 \in \mathcal{L}_1$:

- $m(L_1^{def}) \equiv L_2^{def}$;
- if $L_1^{def} \sqsubseteq_{A'} L_1$, then $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$;
- if $L_1^{def} \sqsubseteq_A L_1$, then $L_1 \equiv L_2$ implies $m(L_1) \equiv m(L_2)$;
- if $L_1^{def} \sqsubseteq_A L_1$ and $L_1^{def} \sqsubseteq_A L_2$, then $m(L_1 \sqcup L_2) \equiv m(L_1) \sqcup m(L_2)$. 🐓

Boolean and evaluation embeddings are incomparable notions. In evaluation embeddings, unlike boolean embeddings, defaults must be mapped to defaults (up to equivalence). Conversely other properties, which recall the characterization of boolean embeddings, are only required to hold for labels that can be put above the defaults using some authority. Compared to *injections* (Definition II.4), evaluation embeddings do not require commutation with meets or with authority-joins. Also, most laws do not have to hold for labels that are not above the defaults. The fact that labels that are not above L^{def} don't matter in the characterization of evaluation embeddings is a direct consequence of the fact that L^{def} is the starting pc label of programs: it is an invariant of the evaluation judgment that labels of results stay above the pc label (using the program's authority).

The proof of the characterization theorem (like its statement) is somewhat intricate because it relies on complex invariants of the evaluation judgment; without the help of a proof assistant, it would be difficult to be confident of its correctness.

Many real world examples of label algebras have a bottom label, used as L^{def}. For that case, the characterization simplifies as follows.

IV.11 Corollary: Assume $\mathcal{M}_1$ and $\mathcal{M}_2$ such that L_1^{def} is a bottom element for $\mathcal{L}_{\mathcal{M}_1}$. Then, $m \in \mathcal{M}_1 \to \mathcal{M}_2$ is an evaluation embedding iff for any A, L_1 and L_2:

- $m(L_1^{def}) \equiv L_2^{def}$;
- $L_1 \sqsubseteq_A L_2$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$;
- $L_1 \equiv L_2$ implies $m(L_1) \equiv m(L_2)$;
- $m(L_1 \sqcup L_2) \equiv m(L_1) \sqcup m(L_2)$. 🐓

In this special case, evaluation embeddings are also boolean embeddings.

IV.12 Proposition: Assume $\mathcal{M}_1 \overset{\text{e}}{\hookrightarrow} \mathcal{M}_2$ and that L_1^{def} is a bottom element for $\mathcal{L}_{\mathcal{M}_1}$. Then $\mathcal{M}_1 \overset{\text{b}}{\hookrightarrow} \mathcal{M}_2$. 🐓

One can wonder under which conditions an evaluation embedding can change the default label. It turns out that if the default label changes from $\bot$ to $\top$ (or from $\top$ to $\bot$), then the input of the embedding must be a trivial label algebra. This result is particularly useful for showing the non-existence of evaluation embeddings.

IV.13 Corollary: Assume $\mathcal{M}_1 \overset{\text{e}}{\hookrightarrow} \mathcal{M}_2$. If $L_1^{def} \equiv \bot$ and $L_2^{def} \equiv \top$, then $L \equiv \bot$ for any $L \in \mathcal{L}_1$. Dually, if $L_1^{def} \equiv \top$ and $L_2^{def} \equiv \bot$, then $L \equiv \top$ for any $L \in \mathcal{L}_1$. 🐓

Consequently, if the default label of a non-degenerate model is $\bot$ or $\top$ then, since dualization changes bottoms to tops (or conversely), there is no evaluation embedding between the model and its dual. However, when the default label is neither a bottom nor a top, an evaluation embedding may exist between a label algebra and its dual. Indeed, for any label algebra $\mathcal{M}$, there is an injection (and thus, an evaluation embedding thanks to theorem IV.7) from $\mathcal{M} \times \mathcal{M}^{op}$ to $(\mathcal{M} \times \mathcal{M}^{op})^{op}$: it suffices to take the map that swaps the components of a pair.

V. Abstract Examples

In this section, we define a number of relatively simple label algebras and investigate embeddings among them. Our goals are twofold: (1) to catalog common examples from the literature, and (2) to expose some interesting symmetries among these examples. In particular, we can define four familiar label models—the *Taint*, *Endorsement*, *Readers*, *Distrust* models—by varying the lattice order and default label of the same basic structure, where labels are sets of principals. We use the characterization theorems to exhaustively settle the existence or non-existence of embeddings among all of these simple models (Figure 4).

UE: Universal Endorsement Model

$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2 \qquad L^{def} = \emptyset$

UT: Universal Taint Model

$L_1 \sqsubseteq_A L_2 = L_1 \subseteq L_2 \cup A \qquad L^{def} = \emptyset$

UR: Universal Readers Model

$L_1 \sqsubseteq_A L_2 = L_1 \cup A \supseteq L_2 \qquad L^{def} = \mathbb{P}$

UD: Universal Distrust Model

$L_1 \sqsubseteq_A L_2 = L_1 \subseteq L_2 \cup A \qquad L^{def} = \mathbb{P}$

Figure 3. Universal models. In this figure, $\mathcal{L} = \mathcal{A} = \mathcal{P}(\mathbb{P})$. Note that **UE** and **UR** have the same flows-to relation. The same is true of **UT** and **UD**.

Universal models. We first consider four *universal* models: the Universal Endorsement, Taint, Readers, and Distrust models. They are "universal" in the sense that a label is an arbitrary—finite or infinite—subset of the whole set of principals. All four can be specified by varying two parameters: the default label and the direction of the flows-to relation (Figure 3). The universal endorsement model **UE** generalizes the endorsement model **CE** from §II: it has more labels and authorities, but behaves the same otherwise. In the universal taint model **UT**, a label represents a set of principals who have tainted some piece of data, and data is untainted by default; authorities are used to remove taint. **UT** differs from **UE** only by the orientation of its flow-to relation: they are duals. The universal readers model **UR** extends **CR** with arbitrary sets of principals as labels and authorities. The universal distrust model **UD** is the same as **UT**, but with a top default label. Labels in **UD** can be interpreted as sets of principals who *dis*trust some piece of data (this terminology was proposed by [10]). The default label is $\mathbb{P}$, meaning that, by default, everyone distrusts everything. Authority is used to remove a principal—i.e. to declare that it trusts some piece of data. **UD** is the dual of **UR**.

Syntactic models. The universal models are not well suited for real languages or systems because they are not *syntactic*—in general, their labels may have no finite representation. However, we define "cut-down" versions of the universal models: **CT**, **CE**, **CR** and **CD** are versions of **UT**, **UE**, **UR** and **UD** where labels and authorities are elements of $\mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\}$—i.e. either finite sets of the full set of principals. Even smaller are **T** and **E**, which are versions of **UT** and **UE** where labels and authorities are just finite sets of principals. (Note that **UR** and **UD** cannot be cut down to finite sets because their default labels are $\mathbb{P}$.)

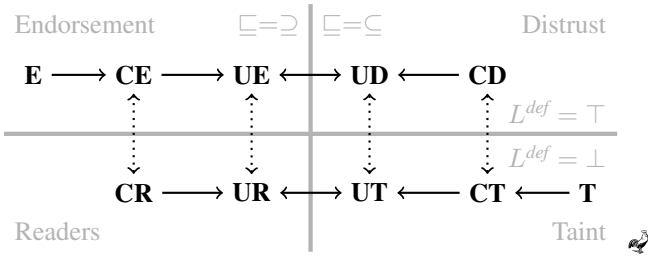

Figure 4. Embeddings among universal and syntactic models. Dotted arrows mean only boolean embeddings.

Embeddings among universal and syntactic models. The relative expressiveness of the universal and syntactic models is summarized in Figure 4. Additionally, we have proved that, for these models, evaluation-embeddability coincides with injectability.

Most of the embeddings are defined using the identity map, and it is very easy to prove that they are indeed embeddings. The evaluation embeddings between universal models, such as the one from **UR** to **UT**, are exceptions, since they are defined by complementation. Proving non-existence of embeddings is significantly harder. Most of the non-existence results are proved either using Corollary IV.13 (for evaluation embeddings between the lower and upper part of the diagram) or else with variants of the following lemma (a consequence of the pigeon-hole principle):

V.1 Lemma: There is no function from $\mathcal{P}^{fin}(\mathbb{P})$ to $\mathcal{P}^{fin}(\mathbb{P})$ that is both bounded and injective.

The proof that there is no boolean embedding from **CT** to **T** is an example of where the above lemma is used: assume such an embedding exists, let us call it $m = (m_{\mathcal{L}}, m_{\mathcal{A}})$. From Theorem IV.8, we know that $m_{\mathcal{L}}$ is necessarily injective on labels (item 2). We also know (item 1) that for any $L \in \mathbf{CT}$, $m_{\mathcal{L}}(L) \subseteq m_{\mathcal{L}}(\mathbb{P}) \cup m_{\mathcal{A}}(\mathbb{P})$. Therefore, the restriction of $m_{\mathcal{L}}$ to finite sets satisfies the hypotheses of lemma V.1—a contradiction.

Fig. 4 permits us to draw three (unsurprising) conclusions. First, restricting models to $\mathcal{P}^{fin}(\mathbb{P})$ or $\mathcal{P}^{fin}(\mathbb{P}) \cup \{\mathbb{P}\}$ leads to strictly less expressive label algebras. Second, changing the default label can be observed by evaluation-embeddings but not by boolean embeddings (as stated by Corollaries IV.13 and IV.13 and by Proposition IV.9, respectively). And third, complementation (i.e. the composition of dualization and complementation of the default label) preserves embeddings between universal models: indeed, the complementation map is an injection.

Apropos integrity. The observation that integrity can be treated as a formal dual to confidentiality goes back at least to Biba [13]. This agrees with the fact that **UE** is the formal dual of **UT** and and **UR** the formal dual of **UD**. Many real-world systems (e.g., [8], [14], [5]) have relied on this

observation to provide unified mechanisms for both.

VI. Real-World Examples

We now turn our attention to formalizing the label models of several existing systems. While some do not perfectly fit the formal structure of label algebras, even partial descriptions in a common framework will hopefully help clarify their similarities and differences. Moreover, we can use these formalizations to study the existence or non-existence of embeddings. We do not settle the question for all pairs of examples, but we do establish several results involving variants of the DLM and DC models.

Disjunction-Category (DC) labels. Disjunction Category labels [3] come from a Haskell security library called LIO [14], part of the HAILS framework for secure web apps [15]. **DC** labels have a secrecy part and an integrity part. We first focus on the secrecy part ($\mathbf{DC_S}$).

Labels of $\mathbf{DC_S}$ are finite boolean formulae in conjunctive normal form, containing no negations—i.e. finite conjunctions of finite disjunctions of principals. We write $\mathcal{F}$ to denote the set of such formulae. The flows-to relation is reverse logical entailment, written $\Leftarrow$. Intuitively, these formulae tell who can observe a piece of labeled data. The $\sqsubseteq$ relation allows us to make conservative approximations about who is allowed to observe. True is the empty conjunction and is the $\bot$ element—it means that any principal can observe the data; False is the empty disjunction and is the $\top$ element.

For example, the $\mathbf{DC_S}$ label $L = p_1 \wedge (p_2 \vee p_3)$ can be read "this data can be read by somebody that has p_1's credentials and either p_2's or p_3's." It flows to $p_1 \wedge p_2$, because somebody that has p_1's and p_2's credentials respects the policy of L.

Authorities are also formulae: $L_1 \sqsubseteq_A L_2$ means that $L_1 \Leftarrow (L_2 \wedge A)$. For example, $L \sqsubseteq_{p_1} p_2 \vee p_3$, because somebody that has either p_2's or p_3's credentials and the ability to use p_1's credentials respects L.

$\mathbf{DC_S}$: DC Labels (secrecy part)

$\mathcal{L} = \mathcal{F} \qquad L^{\mathit{def}} = \mathit{True} \qquad \mathcal{A} = \mathcal{F} \qquad 0 = \mathit{True}$
$A_1 \leq A_2 = A_1 \Leftarrow A_2 \qquad A_1 \vee A_2 = A_1 \wedge A_2$
$L_1 \sqsubseteq_A L_2 = L_1 \Leftarrow (L_2 \wedge A)$
$L_1 \sqcup L_2 = L_1 \wedge L_2 \qquad L_1 \sqcap L_2 = L_1 \vee L_2$

(Formulae are kept in normal form, so, strictly speaking, the definitions of join and meet are up to renormalization.)

The full **DC** model adds an integrity component that is the dual of $\mathbf{DC_S}$.

DC: Full DC Labels (LIO)

$\mathcal{L} = \mathcal{F} \times \mathcal{F} \qquad L^{\mathit{def}} = (\mathit{True}, \mathit{True})$
$\mathcal{A} = \mathcal{F} \qquad 0 = \mathit{True}$
$A_1 \leq A_2 = A_1 \Leftarrow A_2 \qquad A_1 \vee A_2 = A_1 \wedge A_2$
$(S_1, I_1) \sqsubseteq_A (S_2, I_2) = S_1 \Leftarrow (S_2 \wedge A)$ and $I_2 \Leftarrow (I_1 \wedge A)$
$(S_1, I_1) \sqcup (S_2, I_2) = (S_1 \wedge S_2, I_1 \vee I_2)$
$(S_1, I_1) \sqcap (S_2, I_2) = (S_1 \vee S_2, I_1 \wedge I_2)$

Note that the default label is neither a top nor a bottom element: it is the pair of the bottom for secrecy and the top for integrity, i.e. the "most public" and the "least endorsed."

Simplified DLM. We next describe a stripped-down version of the Decentralized Label Model [2]: we focus on its secrecy part only, and we defer modeling its principal hierarchy (*acts-for* relation) until §VII. Labels in $\mathbf{DLM_S}$ are sets of policies, where policies are drawn from the set $Pol = \{p \to P \mid p \in \mathbb{P},\ P \in \mathcal{P}^{\mathit{fin}}(\mathbb{P})\}$. The sets on the right hand side of the arrow are called *reader sets*; they are akin to labels of **CR** in that they decrease as we go up in the lattice of labels. For instance, the label $L_1 = \{p \to \{p_1, p_2\}\}$ says that the principal p allows only principals p_1 and p_2 to read some data. Label $L_2 = \{p \to \{p_1\}\}$ is strictly more secure than L_1—i.e. $L_1 \sqsubseteq L_2$—since L_2 allows fewer possible readers.

When $(p \to P) \in L$, we say that principal p *owns* the policy $p \to P$ in L. This label model is called *decentralized* because several principals can independently own different policies on the same data. In $L_3 = \{p \to \{p_1, p_2\}, q \to \{p_1\}\}$, principals p and q each express a policy. The resulting policy is the *intersection* of p's and q's policies—i.e., both of their policies must be enforced. Note that $L_1 \sqsubseteq L_3$ and that $\{q \to \{p_1\}\} \sqsubseteq L_3$.

Authorities are sets of *owners*. They specify which policies can be modified: for any p in the authority, we can arbitrarily change or remove policies that are owned by p, but other policies can only be changed to more restrictive ones. For instance, $L_3 \sqsubseteq_{\{p\}} \{q \to \{p_1\}\}$ and $L_3 \sqsubseteq_{\{p\}} \{p \to \{p_1, p_2, p_3\}, q \to \{p_1\}\}$. However, $L_3 \not\sqsubseteq_{\{p\}} \{q \to \{p_1, p_2\}\}$, because in the latter label, the security policy owned by q is more permissive than the one in L_3.

$\mathbf{DLM_S}$: Simplified DLM (secrecy, no acts-for)

$\mathcal{L} = \mathcal{P}^{\mathit{fin}}(\mathit{Pol}) \qquad L^{\mathit{def}} = \emptyset \qquad \mathcal{A} = \mathcal{P}^{\mathit{fin}}(\mathbb{P}) \qquad 0 = \emptyset$
$L_1 \sqcup L_2 = L_1 \cup L_2$
$L_1 \sqcap L_2 = \{p \to P_1 \cup P_2 \mid (p \to P_1) \in L_1, (p \to P_2) \in L_2\}$
$L_1 \sqsubseteq L_2 = \forall (p \to P_1) \in L_1.\ \exists (p \to P_2) \in L_2.\ P_2 \subseteq P_1$
$L_1 \sqsubseteq_A L_2 = L_1 \sqsubseteq L_2 \sqcup L_A$ where $L_A = \{p \to \emptyset \mid p \in A\}$

Note that a given principal can own more than one policy, and that this is different from owning a compound policy: $L_1 \sqsubseteq \{p \to \{p_1\}, p \to \{p_2\}\}$, but not conversely. Interestingly, the intuition based on readers sets does not extend to the case of principals owning several policies. For instance, one could expect that $\{p \to \{p_1\}, p \to \{p_2\}\}$ and $\{p \to \emptyset\}$ express the same requirement, i.e. "p says that nobody can read the data". However, the former label is *strictly* lesser than the latter. Here is how we understand such labels: the sets of the right hand side express disjunctions of principals, whereas the juxtaposition of two policies means their conjunction. We conjecture that one can isomorphically

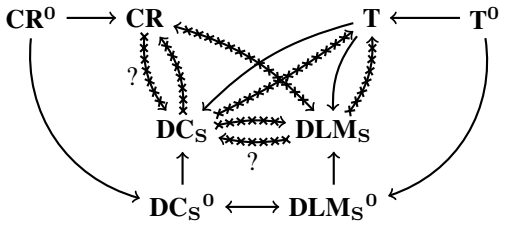

Figure 5. Embeddings with $\mathbf{DLM_S}$ and $\mathbf{DC_S}$. Plain arrows are boolean embeddings; crossed arrows denote non-existence of boolean embeddings; '?' means conjecture.

represent $\mathbf{DLM_S}$ labels as finite maps from principals to $\mathcal{F}$ (conjunctions of disjunctions of principals).

Li *et al.* [10] state (informally) that the two point model, the writer model, the endorsement model and the distrust model can all be encoded in the DLM. Unfortunately, while they describe the lattice structures of all the models they consider, they do not specify their default labels, and they ignore authority. We will see in the next section that authorities play an interesting role.

Some embeddability results. Figure 5 gathers some boolean embeddability results involving $\mathbf{DC_S}$, $\mathbf{DLM_S}$, and some of the models from §V. Note that this table represents ongoing work: we have not yet carried out an exhaustive exploration of this area.

The first thing to notice is that neither **CR** nor **T** are expressive enough to express $\mathbf{DC_S}$ of $\mathbf{DLM_S}$ labels (this is no surprise). More interesting is that the presence of authorities sometimes *precludes* embeddability. For instance, $\mathbf{DC_S}^\mathbf{0}$ and $\mathbf{DLM_S}^\mathbf{0}$, which don't have authorities, embed in each other, but this is not true for their authority-enriched versions (we have proved one way, conjectured the other): intuitively, there is no notion of *owner* of a policy in $\mathbf{DC_S}$, while conversely there is no way to form a disjunction of authorities in $\mathbf{DLM_S}$. Another instance of this phenomenon is that **CR** does not embed in $\mathbf{DC_S}$ or $\mathbf{DLM_S}$, but its 0-authority version does. By contrast, the behavior of **T** is not influenced by the presence of authorities. For the sake of concreteness, we detail one embedding and one non-embeddability arrow of Figure 5.

VI.1 Proposition: $\mathbf{T} \overset{\text{e}}{\hookrightarrow} \mathbf{DLM_S}$.

Proof: Define $m(S) = \bigcup_{p\in S}\{p \to \emptyset\}$ and $m(A) = A$. The map m verifies the conditions of Corollary IV.11. □

VI.2 Proposition: **CR** does not boolean embed in $\mathbf{DLM_S}$.

Proof: Assume that such a boolean embedding exists; let's call it m.

Let us first prove (1): there exists A_0 such that for any label $L \in \mathbf{DC_S}$, $\mathrm{dom}\, m(L) \subseteq A_0$. Take $A_0 = m(0) \cup \mathrm{dom}\, m(\emptyset)$. For any label $L \in \mathbf{CR}$, $L \sqsubseteq_0 \emptyset$, therefore $m(L) \sqsubseteq_{m(0)} m(\emptyset)$ by Theorem IV.8. Then, $\mathrm{dom}\, m(L) \subseteq A_0$ by definition.

Then, let us show (2): for any $A \in \mathcal{A}_\mathbf{CR}$ and $L_1, L_2 \in \mathcal{L}_\mathbf{CR}$, $m(L_1) \sqsubseteq_{m(A)\cap A_0} m(L_2)$ iff $m(L_1) \sqsubseteq_{m(A)} m(L_2)$. The direct way holds by properties of label algebras, since $m(A) \cap A_0 \leq m(A)$. Let us show the converse: assume $m(L_1) \sqsubseteq_{m(A)} m(L_2)$. Assume $p \to P_1 \in m(L_1)$. If $p \in m(A) \cap A_0$, then $(p \to \emptyset) \in m(L_2) \sqcup L_{m(A)\cap A_0}$, which concludes the proof. Assume now, that $p \notin m(A) \cap A_0$. We know that $p \in A_0$ by (1), thus $p \notin m(A)$. Therefore, there exists P_2 such that $p \to P_2 \in m(L_2)$ and $P_2 \subseteq P_1$, by definition of $\sqsubseteq$ in $\mathbf{DLM_S}$. Then, $p \to P_2 \in m(L_2) \sqcup L_{m(A)\cap A_0}$, which concludes the proof that $m(L_1) \sqsubseteq_{m(A)\cap A_0} m(L_2)$.

Let us consider the function $f = \lambda A.\ A \cap A_0$. Let us show that f is injective. Assume that $m(A_1) \cap A_0 = m(A_2) \cap A_0$ (3). Then, for any L_1 and L_2:

	$L_1 \sqsubseteq_{A_1} L_2$	
iff	$m(L_1) \sqsubseteq_{m(A_1)} m(L_2)$	by Theorem IV.8
iff	$m(L_1) \sqsubseteq_{m(A_1)\cap A_0} m(L_2)$	by (2)
iff	$m(L_1) \sqsubseteq_{m(A_2)\cap A_0} m(L_2)$	by (3)
iff	$m(L_1) \sqsubseteq_{m(A_2)} m(L_2)$	by (2)
iff	$L_1 \sqsubseteq_{A_2} L_2$	by Theorem IV.8.

Then, since $\emptyset \sqsubseteq_{A_2} A_2$, we have $\emptyset \sqsubseteq_{A_1} A_2$ (by taking $L_1 = \emptyset$ and $L_2 = A_2$), i.e. $A_2 \subseteq A_1$. Similarly, since $\emptyset \sqsubseteq_{A_1} A_1$, we have $\emptyset \sqsubseteq_{A_2} A_1$, i.e. $A_1 \subseteq A_2$. We proved $A_1 = A_2$. Thus, f is injective. The function f is also bounded by A_0, which contradicts Lemma V.1. Therefore, the embedding m cannot exist. □

These examples show that authorities should be included in discussions of encodability between label models, as they can lead to surprising results. And informal claims about the expressive power of label models really need to be taken with a grain of salt!

Asbestos. Asbestos [4] is a high-security operating system based on information flow. Its labels are maps from principals to security levels. *Level* is the set $\{\star, 0, 1, 2, 3\}$ equipped with the total order $\star < 0 < 1 < 2 < 3$. Labels are composed of a finite map from principals to levels, plus a default level for the principals ("*categories*") that are not mentioned in the map. If $L = (f, l) \in (\mathbb{P} \overset{\text{fin}}{\to} \mathit{Level}) \times \mathit{Level}$, let $L(p)$ denote $f(p)$ when $p \in \mathrm{dom}\, f$ and the default level l otherwise. The ordering on labels is the pointwise extension of the level ordering.

Each Asbestos process owns a set of "privileges," i.e., a set of principals: if p is in that set, a process is allowed to freely change the level owned by p in a label L, à la **DLM**.

Asbestos:

$\mathcal{L} = (\mathbb{P} \overset{\text{fin}}{\to} \mathit{Level}) \times \mathit{Level} \qquad L^{\mathit{def}} = (\{\}, 1)$
$\mathcal{A} = \mathcal{P}^{\mathit{fin}}(\mathbb{P}) \qquad 0 = \emptyset$
$L_1 \sqsubseteq_A L_2 = \forall p, p \in A \text{ or } L_1(p) \leq L_2(p)$

Early HiStar. Different descriptions of the HiStar operating system propose somewhat different notions of labels. The earliest version [5] uses labels inspired by Asbestos, but with several differences in the way they are used—e.g., thread label changes are required to be explicitly stated. In the rest of the section, we focus on differences with respect to labels.

The main difference is the way untainting (reclassification) is handled. In HiStar, the level $\star$ is a privileged level that can only appear in thread labels (as opposed to other kernel objects, such as files), where it confers the right to untaint a principal. It is low in the lattice (below the default level) so that threads need authority to gain untainting privileges.

When a thread with label L_T attempts to observe an object with label L_O, common information-flow rules would require that $L_O \sqsubseteq L_T$. However, that would not correspond to $\star$ being an untainting privilege, since its low position in the lattice prevents flows instead of allowing more flows. The authors explain that the meaning of $\star$ is either bottom or top, depending on the situation. For that purpose, they introduce a special level ✪ (*high star*) that behaves like a maximum level, but is not really part of the lattice of levels: "level ✪ is only used in access rules and never appears in labels of actual objects". Now, the actual read check is $L_O \sqsubseteq L_T^{✪}$, where $L_T^{✪}$ is the label L_T in which every occurrence of $\star$ is replaced with the special top element ✪.

As it stands, HiStar does not fit the label algebra interface, though we conjecture and it is possible to recast the definitions so that labels do form a label algebra. Indeed, the fact that $\star$ occurs in a thread label is a way to express the privileges that are owned by that thread. We can define $Auth(L) = \{p \mid L(p) = \star\}$: it is the authority of a thread that has L as a thread label. Then, under the assumption that $\star$ does not occur in L_1, $L_1 \sqsubseteq L_2^{✪}$ is equivalent to $L_1 \sqsubseteq_{Auth(L_2)} L_2$, where the indexed flows-to relation is the one of Asbestos. (Note that our assumption about L_1 makes sense, since L_1 is supposed not to be a thread label.) We leave the details of this reconstruction of HiStar to future work (but it probably does not warrant very high priority, since HiStar's original label model was in any case subsequently abandoned by its designers).

Later HiStar & DStar. The DStar system [11] and the more recent version of HiStar [6] use a much simpler form of labels that does fit our framework. Their labels are pairs of finite sets of principals (one for secrecy, one for integrity). Indeed, the only difference from $\mathbf{T} \times \mathbf{E}$ is that DStar's set of secrecy principals is disjoint from its set of integrity principals.

Flume & Laminar. In the Flume [8] and Laminar [7] information-flow operating systems, labels are pairs of sets of principals ("*tags*"), one for secrecy, the other for integrity. Like the latest HiStar, these labels are essentially the product of the taint model with the endorsement model.

However, Flume has a notion of authority that makes the description above inaccurate: authorities are sets of "capabilities", which are principals equipped with a *polarity* annotation, positive or negative. A positive capability permits adding a principal to a label (in whichever component), whereas a negative capability allows removing a principal. That behavior of authorities is captured in Figure 6.

Flume: not a label algebra

$$\mathcal{L} = \mathcal{P}^{fin}(\mathbb{P}) \times \mathcal{P}^{fin}(\mathbb{P}) \qquad L^{def} = (\emptyset, \emptyset)$$
$$(S_1, I_1) \sqsubseteq (S_2, I_2) = S_1 \subseteq S_2 \text{ and } I_1 \supseteq I_2$$

$$\mathcal{A} = \mathcal{P}^{fin}(\mathbb{P}) \times \mathcal{P}^{fin}(\mathbb{P}) \qquad 0 = (\emptyset, \emptyset)$$
$$(C_1^+, C_1^-) \leq (C_2^+, C_2^-) = C_1^+ \subseteq C_2^+ \text{ and } C_1^- \subseteq C_2^-$$

$$(S_1, I_1) \sqsubseteq_{(C^+, C^-)} (S_2, I_2) =$$
$$S_2 \setminus S_1 \subseteq C^+ \text{ and } S_1 \setminus S_2 \subseteq C^-$$
$$\text{and } I_2 \setminus I_1 \subseteq C^+ \text{ and } I_1 \setminus I_2 \subseteq C^-$$

Figure 6. Flume labels and authorities.

The definition does not form a label algebra, because using the least authority is not the same as using no authority: $\sqsubseteq_0$ is the equality relation on labels, which is different from the relation $\sqsubseteq$ of the underlying lattice. This refinement makes Flume more flexible: the way thread labels can change is completely programmable, which gives programmers a lot of freedom to define the shape of the lattice. In practice, we don't know whether programmers actually used all this flexibility. If not, we can easily restrict Flume to yield a label algebra by using disjoint principals for confidentiality and integrity and by always giving threads the p^+ authority for confidentiality principals p and the q^- capability for endorsement principals q.

VII. Principal hierarchies

Several information-flow systems (e.g., Jif [2] and Aeolus [16]) allow users to define a hierarchy of principals, often called an *acts-for* relation, which can be used to delegate authority between principals and to implement authorization groups. Because the goal of principal hierarchies is to relax the rules of how information can flow, they can be viewed as part of the authority structure of a label algebra.

Formally, a principal hierarchy is a partial order over principals. Let $\mathcal{H}$ denote the set of all such partial orders. When $H \in \mathcal{H}$ and $(p_1, p_2) \in H$, we say "p_2 *acts for* p_1 (under $\mathcal{H}$)." The bottom principal hierarchy is $H_0 = \{(p, p) \mid p \in \mathbb{P}\}$, because by default, each principal acts for himself. We also need to add a top element to $\mathcal{H}$, because label algebras require a join on authorities: without that top element, it would be impossible to fulfill that requirement. Indeed, "joining" two partial orders leads in general to a preorder: cycles can appear. Here is an example of that phenomenon: suppose that p acts for q in H_1, and that q acts for p in H_2. If $H_1 \vee H_2$ existed as a partial order, then p would necessarily act for q under $H_1 \vee H_2$ and *vice versa*—a contradiction. Adding a top

$\mathbf{DLM_S}$-H: not a label algebra

$\mathcal{L} = \mathcal{P}^{fin}(Pol) \qquad \mathcal{A} = \{\top\} + (\mathcal{H} \times \mathcal{P}^{fin}(\mathbb{P}))$
$0 = (H_0, \emptyset)$
$(H, P) \leq \top = \text{true}$
$(H_1, P_1) \leq (H_2, P_2) = H_1 \subseteq H_2 \text{ and } P_1 \subseteq P_2$
$(H_1, P_1) \vee (H_2, P_2) = ((H_1 \cup H_2)^+, P_1 \cup P_2)$
if $(H_1 \cup H_2)^+$ is a partial order
$A_1 \vee A_2 = \top$ otherwise
$L_1 \sqsubseteq_\top L_2 = \text{true}$
$L_1 \sqsubseteq_{(H,\emptyset)} L_2 = \forall (p_1 \to P_1) \in L_1,\ \exists (p_2 \to P_2) \in L_2,$
$(p_1, p_2) \in H$ and
$\forall p_2' \in P_2, \exists p_1' \in P_1, (p_1', p_2') \in H$
$L_1 \sqsubseteq_{(H,A)} L_2 =$
$L_1 \sqsubseteq_{(H,\emptyset)} L_2 \sqcup L_A$ where $L_A = \{p \to \emptyset \mid p \in A\}$

Figure 7. $\mathbf{DLM_S}$ with principal hierarchy.

element is a technical expedient that allows us to turn a partial join into a total one: top can be understood as an "invalid" principal hierarchy.

To illustrate how this might work, Figure 7 gives a definition of $\mathbf{DLM_S}$-H—the extension of $\mathbf{DLM_S}$ with principal hierarchies. Authorities are pairs of a hierarchy and a set of principals; as in $\mathbf{DLM_S}$, this set is used to remove policies from labels. The relation $\sqsubseteq_{(H_0,A)}$ of $\mathbf{DLM_S}$-H is exactly the relation $\sqsubseteq_A$ of $\mathbf{DLM_S}$.

However, $\mathbf{DLM_S}$-H is *not* a label algebra, because the $\sqcap$ operation on labels does not necessarily define the *greatest* lower bound of labels. (This was already noted by Myers [17].)

We can add principal hierarchies to other label algebras in a similar way, leading to similar problems in most instances. The following table sums up the effect of that change on the joins and meets of some of the label algebras we've discussed. The checkmark ✓ indicates that joins (resp. meets) remain least upper bounds (greatest lower bounds); the cross ✗ denotes that joins (meets) are not the best upper bound (lower bound).

Adding acts-for to...	**T**	**R**	**D**	**E**	$\mathbf{DC_S}$	$\mathbf{DLM_S}$
$\sqcup$ still works	✓	✗	✓	✗	✓	✓
$\sqcap$ still works	✗	✓	✗	✓	✓	✗

Interestingly, none of the set-based models of §V extends well with principal hierarchies (some of them lose joins, others lose meets), whereas $\mathbf{DC_S}$ presents no problem. The reason is that set intersection loses some information, whereas syntactic meets—i.e. disjunctions in the case of $\mathbf{DC_S}$—keep all the available information. (Take, for example, a hierarchy H such that p acts for q in H. In **T**, $\{p\} \sqcap \{q\} = \{p\} \cap \{q\} = \emptyset$, although $\{q\} \sqsubseteq_H \{p\}$, so a more precise lower bound for $\{p\}$ and $\{q\}$ is $\{q\}$ when they are considered under H.)

Fortunately, there is a generic way to recover the joins or meets that disappeared while adding principal hierarchies. Given a preorder $(X, \sqsubseteq)$, we can define the *join-* and *meet-completion* of $\sqsubseteq$, written $\sqsubseteq^\sqcup$ and $\sqsubseteq^\sqcap$. These restore the accuracy of joins or meets by adding new points in the preorder structure as needed. Informally, the meet-completed preorder introduces new points that represent a syntactic n-ary meets of points. Formally, $\sqsubseteq^\sqcap$ is the relation on $\mathcal{P}^{fin}(X)$ such that $L_1 \sqsubseteq^\sqcap L_2 = \forall l_2 \in L_2, \exists l_1 \in L_1, l_1 \sqsubseteq l_2$. Meet is just set union: $\{l_1, \ldots, l_n\}$ literally represents the meet of $l_1, \ldots, l_n$. If X has joins, then the join in the meet-completed preorder is defined as follows: $L_1 \sqcup^\sqcap L_2 = \{l_1 \sqcup l_2 \mid l_1 \in L_1, l_2 \in L_2\}$. Meet-completion does not change the accuracy of joins: for L_1 and L_2 in X, $\{L_1\} \sqcup^\sqcap \{L_2\} = \{L_1 \sqcup L_2\}$.

The definition of join-completion is dual to the one of meet-completion: join-completion adds syntactic n-ary joins to the set of points. The definitions of join- and meet-completions can be found in the Appendix.

Adding acts-for to **R** and then performing join-completion leads to a label algebra. Similarly, meet-completing $\mathbf{DLM_S}$-H gives back a label algebra.

Interestingly, $\mathbf{T}^\sqcap$ and $\mathbf{R}^\sqcup$ are isomorphic to $\mathbf{DC_S}$: the **DC** label model seems to be the simplest model that is an extension of the set-based models and that supports the addition of principal hierarchies.

VIII. Related Work

Sabelfeld and Sands [18] describe some aspects of declassification and what rules it should follow. One of them is *conservativity*: "Security for programs with no declassification is equivalent to noninterference". This rule corresponds to one of the axioms of label algebras, namely: $\sqsubseteq = \sqsubseteq_0$, i.e. the bottom authority plays no role. Instantiating Theorem III.3 with the 0-authority gives indeed noninterference for programs that do not declassify.

Mantel and Sands [19] study *intransitive non-interference*, a security property of programs that perform declassification. They use PERs that rely on bisimulations of labeled transition systems: by definition, a *strongly secure* program is related to itself. This seems akin to our evaluation semantics, although they use a more intensional PER on programs. Like our λ-calculus (§III), their language identifies which parts of the code perform declassification. Their small-step semantics is labeled with the authority that is used during reduction. (Keeping such a trace and using it in the evaluation semantics of labels is an interesting direction for future work for us.) Label algebras are not well suited to model intransitive policies, since the ordering on labels is required to be transitive. If we consider the transitive closures of the relations used by the authors, there seems to be only two authorities—no authority ($\bot$) and declassification authority ($\top$). The declassification relation ($\rightsquigarrow$), which allows exceptional flows, is not required to be transitive, and neither is $\sqsubseteq_\bot \cup \rightsquigarrow$ in general. In our notation, $\sqsubseteq_\top$ corresponds to the relation $(\sqsubseteq \cup \rightsquigarrow)^+$.

Paralocks [20] is a language for building statically verifiable information-flow policies. It is based on locks that are guarded by policies in the form of Horn clauses, which form a pre-lattice. The authors prove that the DLM can be encoded into Paralocks by defining a function on labels that preserves and reflects the flows-to relation and commutes with joins and meets: ignoring authority and the default label, this is a label algebra injection. It is not clear whether they take authority into account.

Jaume [21] compares the security policies induced by access control and by information flow analysis. For that purpose, he uses techniques that are similar to the ones we use. A security policy $\mathbb{P}$ is a triple of a set of security targets $\mathbb{T}$, a set of configurations $\mathcal{C}$, and a predicate $\models \subseteq \mathcal{C} \times \mathbb{T}$, where $c \models t$ means that the target t is secure with respect to the configuration c. Assuming the policies $\mathbb{P}_1$ and $\mathbb{P}_2$ are defined over the same set of targets, $\mathbb{P}_1$ can be embedded into $\mathbb{P}_2$ when $\forall c_1 \in \mathcal{C}_1, \exists c_2 \in \mathcal{C}_2, \forall t \in \mathbb{T}, c_1 \models_1 t \Leftrightarrow c_2 \models_2 t$. Policies over different sets of targets are studied by interpreting them into policies over the same targets.

IX. Future Work

This exploration of label algebras and embeddings is just a beginning. Ultimately, we would like to build an exhaustive map of all known label algebras along with the existence or absence of embeddings between every pair. However, even for the part of this map that we've completed so far, using a proof assistant has been more than just helpful—it was actually a necessity. For one thing, it helped us to come up with cleaner definitions (as formalization always does). More importantly, without machine checking we would not have trusted our proof of Theorem IV.10, nor would we probably ever have been confident in the correctness of Figure 4. Formally verifying further embeddability results—especially among real-world label algebras—is a natural next step.

Since it depends on the semantics of a programming language, the evaluation semantics from §IV is sensitive to the set of features of the language. It would be interesting to extend our language with first-class labels and authorities, and possibly imperative features, and see the implications on the characterization Theorem IV.10. We conjecture that extending the language in a way that makes use of more features of label algebras leads to embeddings that need to preserve and reflect more aspects of label algebras.

We would like also to experiment with different instances of the abstract label semantics: for example, our evaluation semantics could use a more semantic notion of program equivalence. We could also consider an information-flow type system for our language (i.e., one parameterized over label algebras) and then use the typability judgment to define a label semantics.

Label algebras cannot currently describe systems that dynamically generate principals (and associated authorities and labels). We have defined an extension that deals with dynamicity, but a full exposition is beyond the scope of this paper. The extension relies on two ingredients: (1) standard techniques from nominal logics (permutation of principals, finite support, equivariance, etc.) for dealing with generation of fresh principal names, and (2) a pair of *reification* and *reflection* functions that define a syntactic representation of labels and authorities. We are currently studying the properties of evaluation embeddings for a language with these features.

Acknowledgments

We are grateful to Steve Zdancewic, David Mazières, Deian Stefan, and the members of the SAFE team for fruitful discussions and to Michael Greenberg for a close reading of an earlier draft. Robin Morisset helped initiate the design of label algebras. Greg Morrisett and Adrien Surée participated in early discussions and gave us essential feedback on the technical definitions. This material is based upon work supported by the DARPA CRASH program through the US Air Force Research Laboratory (AFRL) under Contract No. FA8650- 10-C-7090. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] D. E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, pp. 236–243, May 1976. [Online]. Available: http://doi.acm.org/10.1145/360051.360056

[2] A. C. Myers and B. Liskov, "Protecting privacy using the decentralized label model," *ACM Trans. Softw. Eng. Methodol.*, vol. 9, pp. 410–442, October 2000. [Online]. Available: http://doi.acm.org/10.1145/363516.363526

[3] D. Stefan, A. Russo, D. Mazières, and J. C. Mitchell, "Disjunction category labels," in *16th Nordic Conference on Secure IT Systems*, ser. NordSec. Springer, 2011, pp. 223–239. [Online]. Available: http://www.scs.stanford.edu/~deian/pubs//stefan:2011:dclabels.pdf

[4] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazières, F. Kaashoek, and R. Morris, "Labels and event processes in the Asbestos operating system," in *20th Symposium on Operating Systems Principles*, ser. SOSP. ACM, 2005, pp. 17–30. [Online]. Available: http://asbestos.cs.ucla.edu/pubs/asbestos-sosp05.pdf

[5] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières, "Making information flow explicit in HiStar," in *Proceedings of the 7th symposium on Operating systems design and implementation*, ser. OSDI. USENIX Association, 2006, pp. 263–278. [Online]. Available: http://www.scs.stanford.edu/~nickolai/papers/zeldovich-histar.pdf

[6] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières, "Making information flow explicit in HiStar," *Communications of the ACM*, vol. 54, no. 11, pp. 93–101, 2011. [Online]. Available: http://www.scs.stanford.edu/~dm/home/papers/zeldovich:histar-cacm.pdf

[7] I. Roy, D. E. Porter, M. D. Bond, K. S. McKinley, and E. Witchel, "Laminar: Practical fine-grained decentralized information flow control," in *Proceedings of the Conference on Programming Language Design and Implementation*, ser. PLDI. ACM, 2009, pp. 63–74. [Online]. Available: http://www.cs.utexas.edu/users/witchel/pubs/roy09pldi.pdf

[8] M. N. Krohn, A. Yip, M. Z. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris, "Information flow control for standard OS abstractions," in *21st Symposium on Operating Systems Principles*, ser. SOSP. ACM, October 2007, pp. 321–334. [Online]. Available: http://pdos.csail.mit.edu/~max/docs/flume.pdf

[9] *The Coq Proof Assistant*, 2012, version 8.4. [Online]. Available: http://coq.inria.fr/refman/

[10] P. Li, Y. Mao, and S. Zdancewic, "Information integrity policies," in *Proceedings of the Workshop on Formal Aspects in Security & Trust (FAST)*, September 2003. [Online]. Available: http://www.cis.upenn.edu/~stevez/papers/LMZ03.pdf

[11] N. Zeldovich, S. Boyd-Wickizer, and D. Mazieres, "Securing distributed systems with information flow control," in *6th Symposium on Networked Systems Design and Implementation*, San Francisco, CA, April 2008. [Online]. Available: http://www.scs.stanford.edu/~dm/home/papers/zeldovich:dstar.pdf

[12] T. H. Austin and C. Flanagan, "Efficient purely-dynamic information flow analysis," *SIGPLAN Notices*, vol. 44, pp. 20–31, December 2009. [Online]. Available: http://slang.soe.ucsc.edu/cormac/papers/plas09.pdf

[13] K. J. Biba, "Integrity considerations for secure computer systems," Mitre, Tech. Rep. ESD-TR-76-372, MTR-3154 Rev 1, Apr. 1977.

[14] D. Stefan, A. Russo, J. C. Mitchell, and D. Mazières, "Flexible dynamic information flow control in Haskell," in *Proceedings of the 4th Symposium on Haskell*. ACM, 2011, pp. 95–106. [Online]. Available: http://www.scs.stanford.edu/~deian/pubs//stefan:2011:flexible-ext.pdf

[15] D. B. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. Mitchell, and A. Russo, "Hails: Protecting data privacy in untrusted web applications," in *10th Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 2012, pp. 47–60. [Online]. Available: http://www.scs.stanford.edu/~deian/pubs//giffin:2012:hails.pdf

[16] W. Cheng, D. R. K. Ports, D. Schultz, J. Cowling, V. Popic, A. Blankstein, D. Curtis, L. Shrira, and B. Liskov, "Abstractions for usable information flow control in Aeolus," in *Proceedings of the 2012 USENIX Annual Technical Conference*, Jun. 2012. [Online]. Available: http://pmg.csail.mit.edu/pubs/cheng12__abstr_usabl_infor_flow_contr_aeolus-abstract.html

[17] A. C. Myers, "Mostly-static decentralized information flow control," Ph.D. dissertation, Massachusetts Institute of Technology, January 1999. [Online]. Available: http://www.cs.cornell.edu/andru/release/tr783.pdf

[18] A. Sabelfeld and D. Sands, "Dimensions and principles of declassification," in *Computer Security Foundations 18th Workshop*, IEEE, Ed., June 2005, pp. 255–269. [Online]. Available: http://www.cse.chalmers.se/~dave/papers/sabelfeld-sands-CSFW05.pdf

[19] H. Mantel and D. Sands, "Controlled declassification based on intransitive noninterference," in *Proc. Asian Symp. on Programming Languages and Systems*, ser. LNCS. Springer-Verlag, 2004, pp. 129–145. [Online]. Available: http://www.cse.chalmers.se/~dave/papers/Mantel-Sands-TR04.pdf

[20] N. Broberg and D. Sands, "Paralocks: role-based information flow control and beyond," *SIGPLAN Not.*, vol. 45, pp. 431–444, January 2010. [Online]. Available: http://www.cse.chalmers.se/~dave/papers/Broberg-Sands-POPL10.pdf

[21] M. Jaume, "Semantic comparison of security policies: from access control policies to flow properties," in *IEEE Workshop on Semantic Computing and Security, WSCS'2012 IEEE CS Security and Privacy Workshops (SPW)*. IEEE Computer Society Press, 2012, pp. 60–67. [Online]. Available: http://www-spi.lip6.fr/~jaume/wscs2012.pdf

APPENDIX

OTHER OPERATIONS ON LABEL ALGEBRAS

A. Meet completion

Meet completion takes a preorder and gives a meet-prelattice. If the preorder has joins, its meet completion preserves them.

$\mathcal{L}^{\sqcap}$: Meet completion of $\mathcal{L}$

$$\begin{aligned}
&\mathcal{L}^{\sqcap} = \mathcal{P}^{fin}(\mathcal{L}) \\
&L_1 \sqsubseteq^{\sqcap} L_2 = \forall l_2 \in L_2, \exists l_1 \in L_1, l_1 \sqsubseteq l_2 \\
&L_1 \sqcup^{\sqcap} L_2 = \{l_1 \sqcup l_2 \mid l_1 \in L_1, l_2 \in L_2\} \\
&L_1 \sqcap^{\sqcap} L_2 = L_1 \cup L_2
\end{aligned}$$

A.1 Lemma: Meet completion improves the accuracy of lower bounds: for any $l, l_1, l_2 \in \mathcal{L}$, such that $l \sqsubseteq l_1$ and $l \sqsubseteq l_2$, we have $\{l\} \sqsubseteq^{\sqcap} \{l_1\} \sqcap^{\sqcap} \{l_2\}$.

A.2 Lemma: Assume $\mathcal{L}$ is a join-prelattice. Meet completion preserves joins: for any $l_1, l_2 \in \mathcal{L}$, $\{l_1 \sqcup l_2\} = \{l_1\} \sqcup^{\sqcap} \{l_2\}$.

B. Join completion

Join completion takes a preorder and gives a join-prelattice. If the preorder has meets, its join completion preserves them.

$\mathcal{L}^{\sqcup}$: Join completion of $\mathcal{L}$

$$\begin{aligned}
&\mathcal{L}^{\sqcup} = \mathcal{P}^{fin}(\mathcal{L}) \\
&L_1 \sqsubseteq^{\sqcup} L_2 = \forall l_1 \in L_1, \exists l_2 \in L_2, l_1 \sqsubseteq l_2 \\
&L_1 \sqcup^{\sqcup} L_2 = L_1 \cup L_2 \\
&L_1 \sqcap^{\sqcup} L_2 = \{l_1 \sqcap l_2 \mid l_1 \in L_1, l_2 \in L_2\}
\end{aligned}$$

B.1 Lemma: Join completion improves the accuracy of upper bounds: for any $l, l_1, l_2 \in \mathcal{L}$, such that $l_1 \sqsubseteq l$ and $l_2 \sqsubseteq l$, we have $\{l_1\} \sqcup^{\sqcup} \{l_2\} \sqsubseteq^{\sqcup} \{l\}$.

B.2 Lemma: Assume $\mathcal{L}$ is a meet-prelattice. Join completion preserves meets: for any $l_1, l_2 \in \mathcal{L}$, $\{l_1 \sqcap l_2\} = \{l_1\} \sqcap^{\sqcup} \{l_2\}$.